Title:	Campaign Storage Implementation and Intentions
Author(s):	Ransom, Garrett Wilson
Intended for:	External Presentation
Issued:	2018-06-05

LA-UR-?

Campaign Storage

Implementation and Intentions

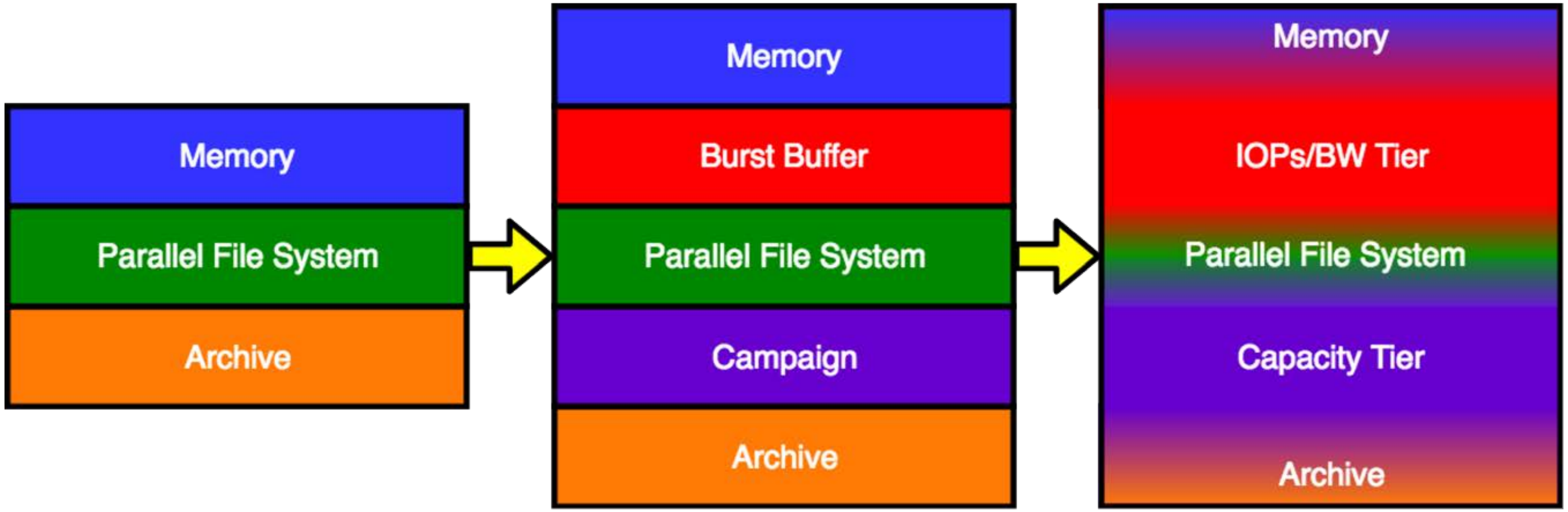
Garrett Ransom (HPC-SYS)

06/04/18

HPC Storage Trends at LANL

- **Data sets are growing faster than long-term storage can support**
  - Trinity: 2PB of memory / 4PB of flash
  - Crossroads: (maybe) 4PB mem / 10-100PB flash
  - HPSS Archive ~60PB Total, continuously expanding
- **Bandwidth of archive is a limiting factor**
  - Usable bandwidth of traditional archive (HPSS) ingest is roughly 3 GB/sec
  - HPSS ingests data much faster than it recalls
  - Storing petabytes of job checkpoints is infeasible

HPC Storage Trends at LANL



Implementing a Capacity Tier

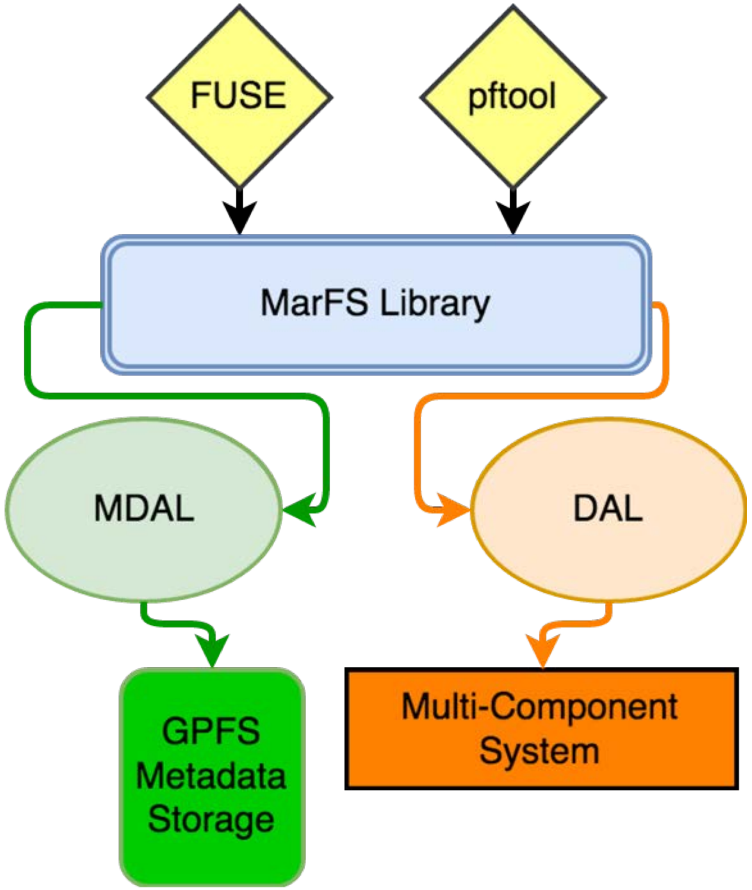
- **Tape is likely not the approach to take**
  - Tape is effective for truly cold data, not data sets that require periodic recall
  - Designing tape storage solutions is complex
- **Object Storage seems promising**
  - Flat namespace allows easy scalability
  - Erasure coding allows for cheaper disk media
- **Object Storage has limitations**
  - Machines love object-IDs, people generally don’t
  - Potentially billions $ in applications expecting ‘POSIX-like’ file trees

What is MarFS?

- **A near-POSIX interface layered over distinct metadata and data implementations**
  - Data stored as erasure coded objects
  - Metadata mirrored within a parallel file system (GPFS)
  - Object IDs written as extended attributes of metadata files
- **Familiar semantics, fast metadata, stable objects**
  - Storing metadata to a real PLFS gives us POSIX-style directory trees and permissions almost for free
  - Storing data as objects simplifies implementation and data protection
- **With tradeoffs, of course**
  - no update in place or file locking
  - restricted interactive use

The Structure of MarFS

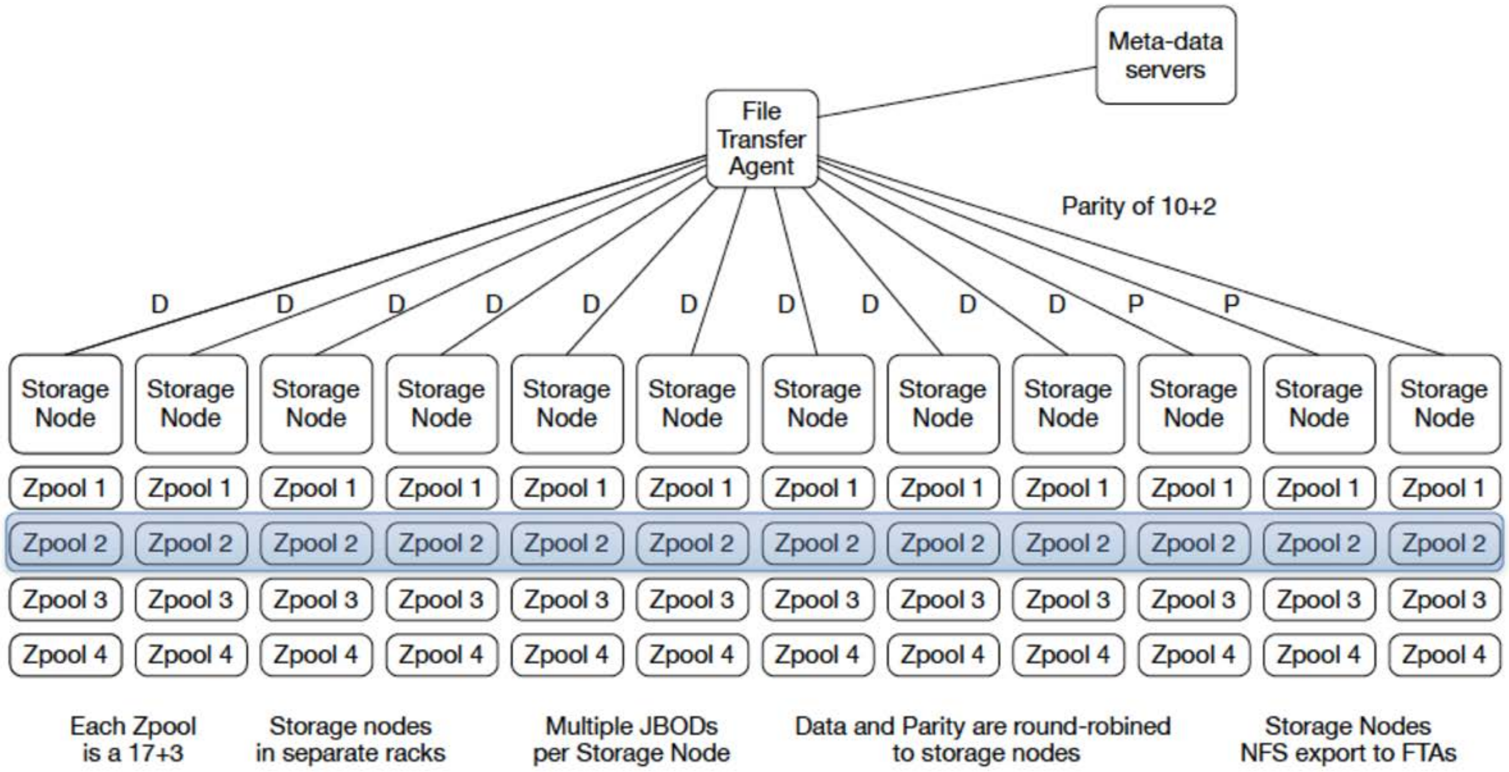
- **Pftool**
  - Parallel MPI file transfer utility
- **FUSE mount**
  - Provides user view of metadata
- **MarFS Library**
  - Heart of the software infrastructure
- **DAL/MDAL**
  - Abstraction layers atop data and metadata respectively
  - Allow easy swapping of underlying storage



Multi-Component

- **Current data storage solution for Campaign**
  - Integrated via the MarFS DAL
  - Stores data as pseudo-objects
- **Cross-server erasure coding atop ZFS pools**
  - Allows failure tolerance at both the disk and server level
  - More reliability allows the use of cheaper disk
  - Erasure coding performed through Intel’s Storage Acceleration Library (isa-l)
- **Performance through parallelism**
  - Simultaneous I/O to multiple servers, each with large arrays of disk

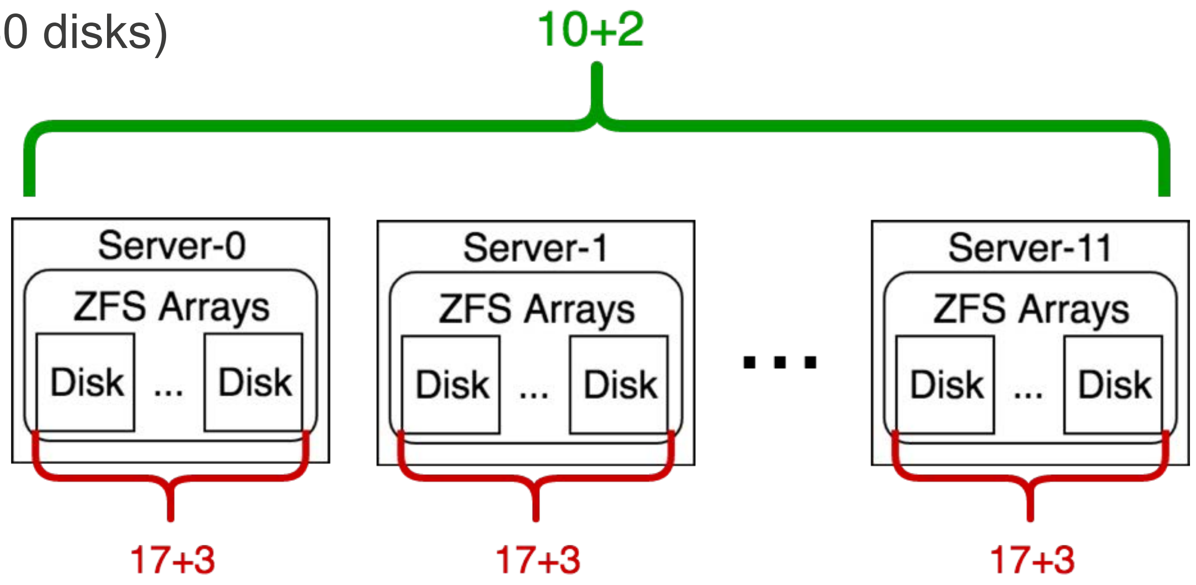
Multi-Component: System Structure


Meta-data servers
File Transfer Agent
Parity of 10+2
D D D D D D D D D D P P
Storage Node
Zpool 1
Zpool 2
Zpool 3
Zpool 4
Each Zpool is a 17+3
Storage nodes in separate racks
Multiple JBODs per Storage Node
Data and Parity are round-robined to storage nodes
Storage Nodes NFS export to FTAs

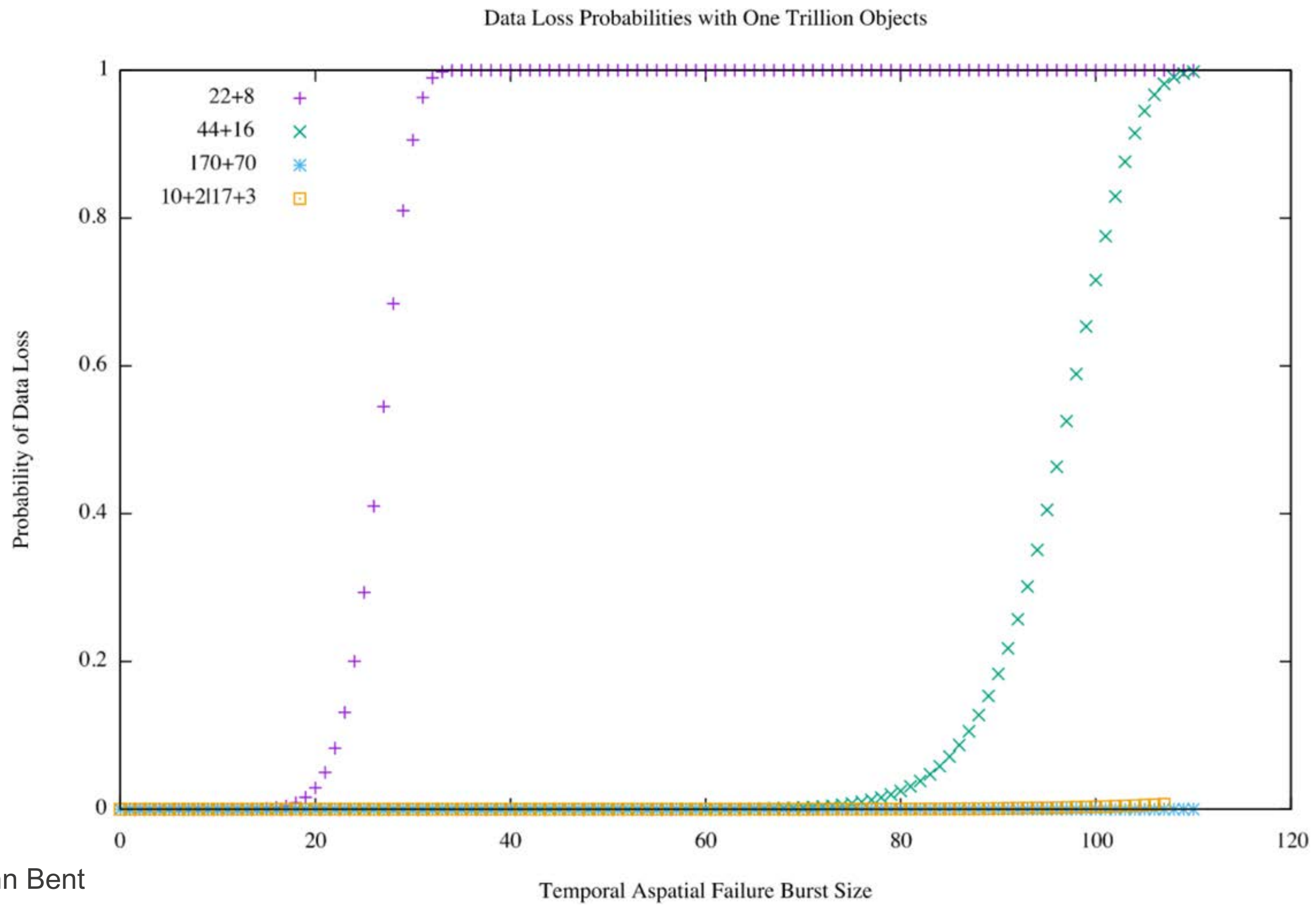
Thanks to Kyle Lamb, Dave Bonnie, and Jeff Inman

Multi-Component: Resiliency

- **Multi-tier erasure**
  - Multi-Component: **10+2** across servers
  - ZFS: **17+3** across disks
  - Tolerates min 11 and max 70 disk failures per stripe (set of 240 disks)



Multi-Component: Resiliency

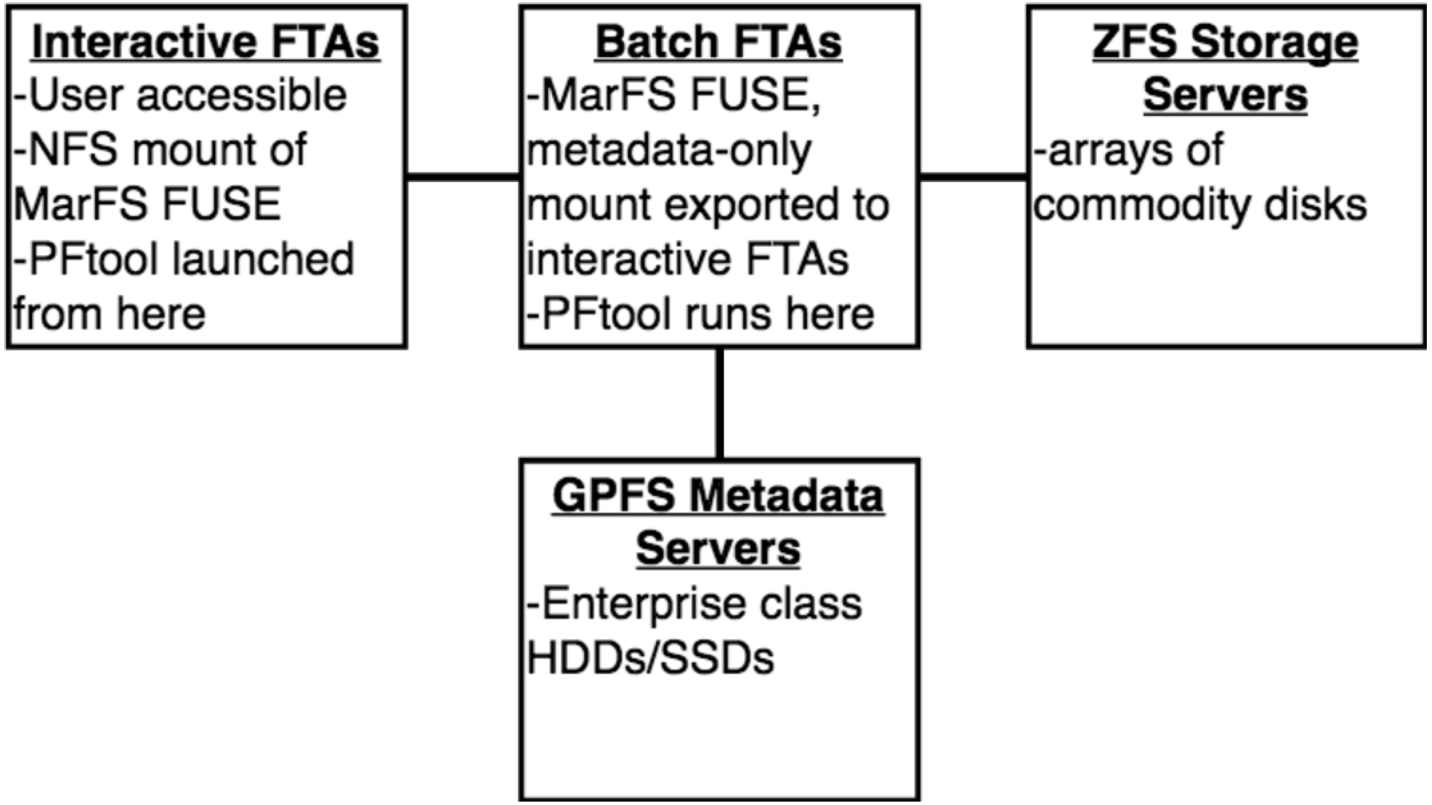


Thanks to John Bent

Multi-Component: Transparency

- **Storing to ZFS systems means that objects are plainly visible to administrators**
  - Each object ‘part’ is paired with a manifest file, providing data and erasure structure info
  - Admins can literally ‘ls’ object parts and ‘cat’ manifest info
- **Utilities exist for interacting directly with objects**
  - Read/write data independent of the entire MarFS stack
  - Object integrity checks
  - Erasure rebuild of damaged objects
- **Standard ZFS features can still be leveraged**
  - Periodic ZFS data scrubs catch silent corruption early

MarFS Multi-Component Deployment


Interactive FTAs
-User accessible
-NFS mount of MarFS FUSE
-PFtool launched from here
Batch FTAs
-MarFS FUSE, metadata-only mount exported to interactive FTAs
-PFtool runs here
ZFS Storage Servers
-arrays of commodity disks
GPFS Metadata Servers
-Enterprise class HDDs/SSDs

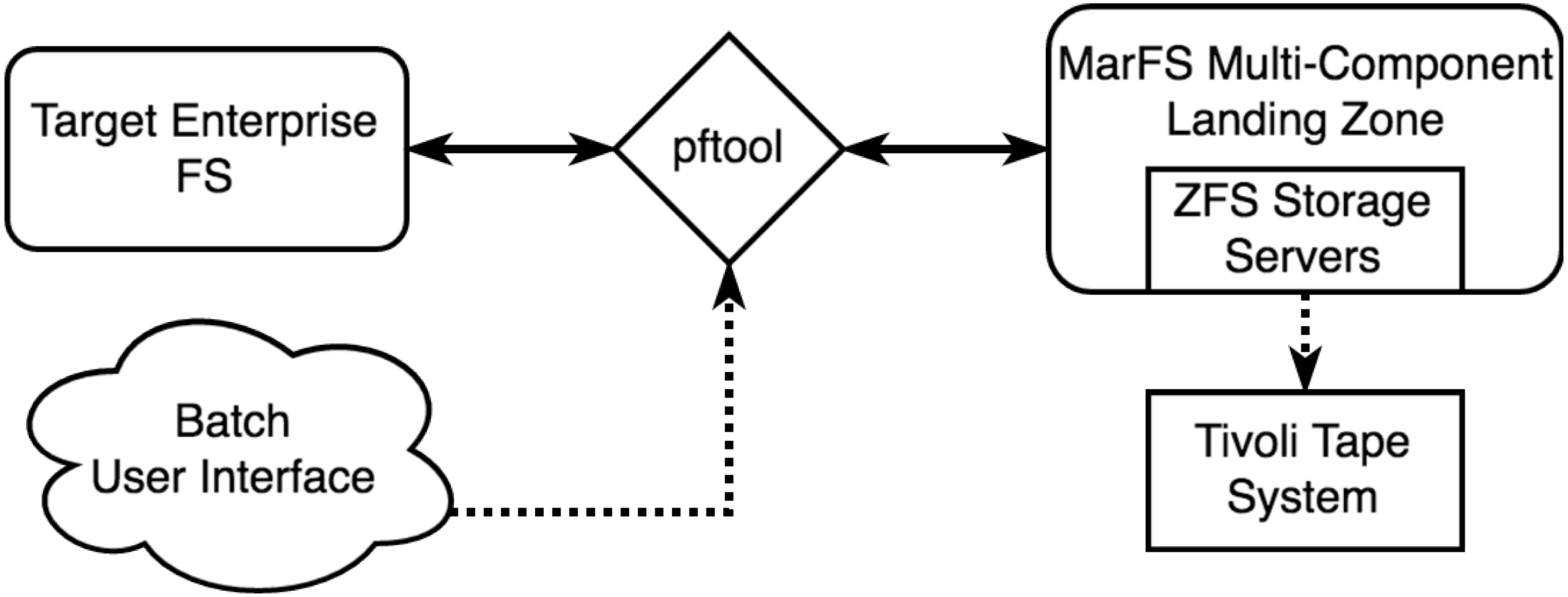
Current Status

- **MarFS Multi-Component has been in production use for two years**
  - 5 interactive and 25 batch FTAs
  - 60PB of total storage
  - Roughly 25 GB/sec aggregate bandwidth for both read and write
  - NFS appears to be the performance bottleneck
- **Currently seeing heavy usage from a subset of users**
  - Largest runs on Trinity necessitate use of Campaign Storage
  - Peak sustained user ingest of approximately 1PB / week
- **Additional deployments in progress for this year**
- **Infiniband RDMA based Multi-Component currently in development**
  - Initial testing yielded 35 GB/sec with a fraction of the mpi ranks

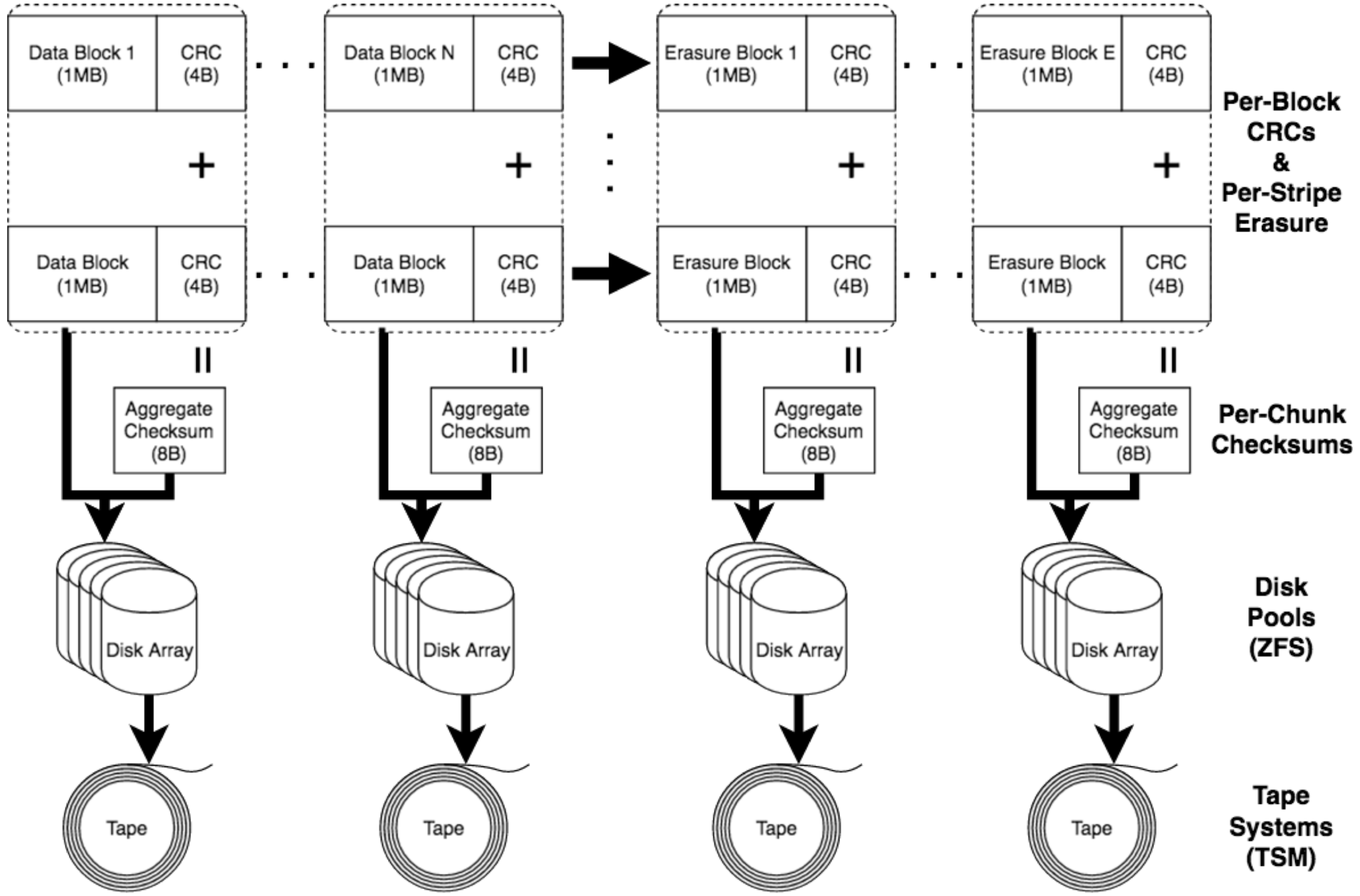
Future Work

- **Better documentation and administrator controls**
- **RDMA data transport**
  - Significant bandwidth improvement may allow for smaller Lustre scratch space in future procurements
- **Capacity-unit migration**
  - Migration of objects to new storage with no downtime
- **Job scheduling for transfers?**
  - Current lack of scheduling means simultaneous transfers always compete for resources
- **Tape-backed MarFS Multi-Component (Marchive)**

Marchive: Conceptual Implementation



Marchive: Data Path

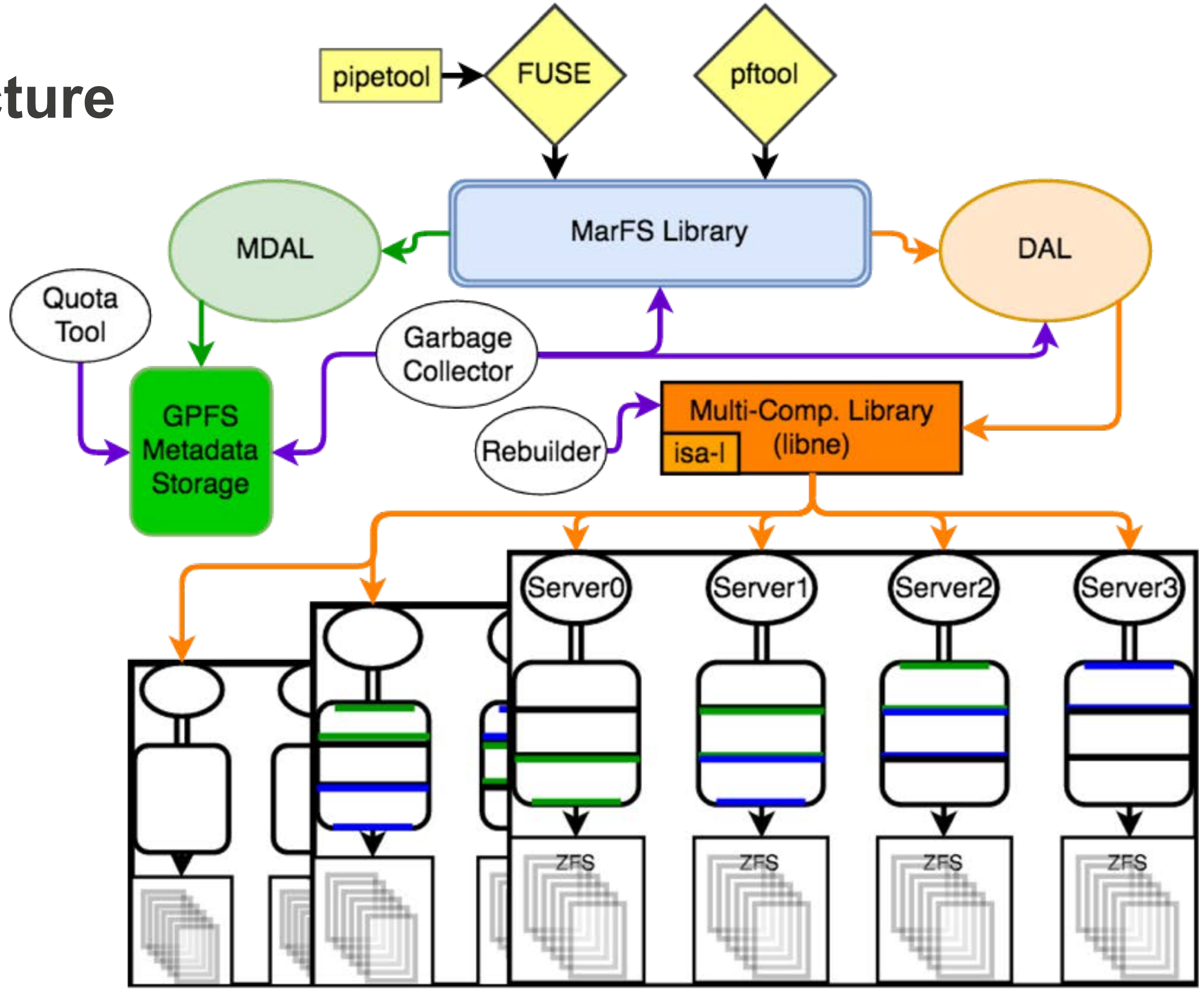


Github Organization – https://github.com/mar-file-system

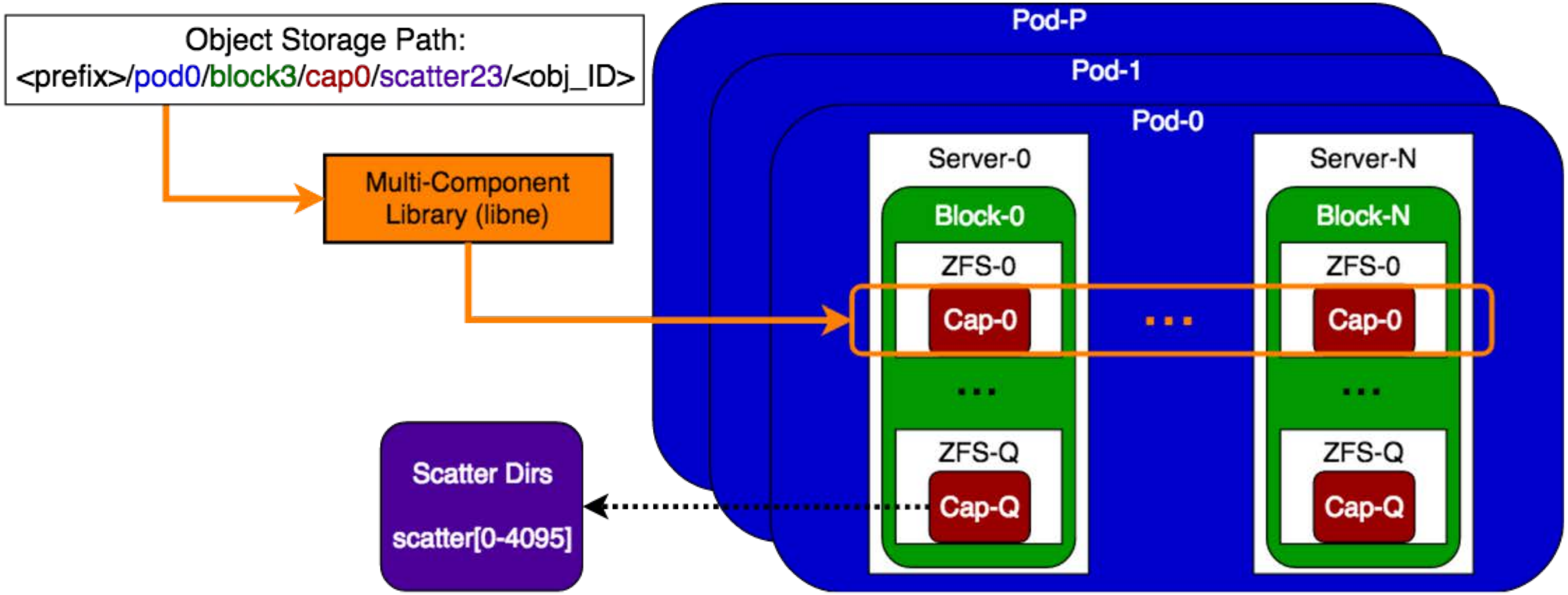
- **MarFS – https://github.com/mar-file-system/marfs**
- **LibNE – https://github.com/mar-file-system/erasureUtils**

Pftool – https://github.com/pftool/pftool

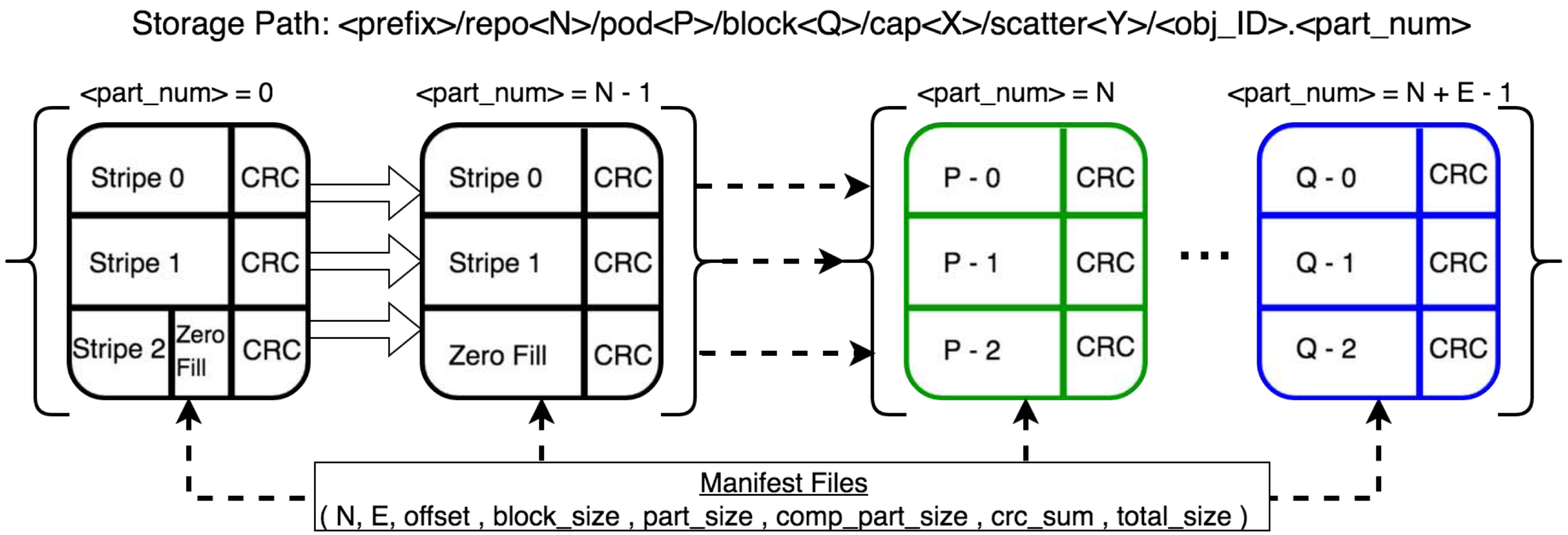
Extra Info: The Big Picture



Extra Info: Multi-Component Object Distribution



Extra Info: Multi-Component Data Stripe Structure


Storage Path: <prefix>/repo<N>/pod<P>/block<Q>/cap<X>/scatter<Y>/<obj_ID>.<part_num>
<part_num> = 0
<part_num> = N - 1
<part_num> = N
<part_num> = N + E - 1
Stripe 0
CRC
Stripe 1
CRC
Stripe 2
Zero Fill
CRC
Stripe 0
CRC
Stripe 1
CRC
Zero Fill
CRC
P - 0
CRC
P - 1
CRC
P - 2
CRC
...
Q - 0
CRC
Q - 1
CRC
Q - 2
CRC
Manifest Files
(N, E, offset , block_size , part_size , comp_part_size , crc_sum , total_size)